Lógica de Programação

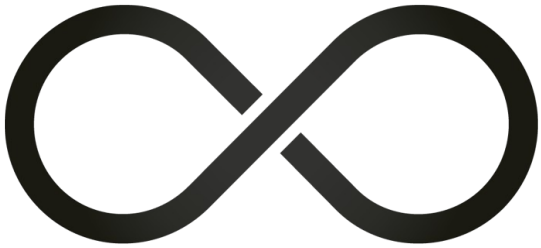
Módulo 04

Estruturas de Repetição

Introdução

Vamos entrar agora num assunto muito interessante. Já ouviu falar de **loop**? Se ainda não ouviu, vá se acostumando com a expressão, pois ouvirá muito durante sua carreira na área de TI.

Loop é uma palavra inglesa, que significa 'laço', 'aro', 'anel', 'circuito' ou 'sequência'. Quando nos referimos a um **loop** dentro de um programa e queremos falar em português, normalmente usamos a palavra **laço**. Em termos de programação o **loop** ou **laço** é um recurso para executar determinadas ações (repetir as ações) até que uma condição seja satisfeita.



Dependendo da linguagem de programação, os laços podem ser implementados de maneiras diferentes. No entanto, existem três variações que costumam ser implementadas por todas elas. Vamos estudar essas três variações a seguir.

1. ENQUANTO-FAÇA
2. REPITA-ATÉ / FAÇA-ENQUANTO
3. PARA-FAÇA

Estrutura de Repetição ENQUANTO-FAÇA

O funcionamento da estrutura de repetição ENQUANTO-FAÇA (em inglês WHILE-DO) é tão simples quanto o SE-ENTÃO-SENÃO. A diferença é que os passos dentro deste bloco são repetidos enquanto a expressão booleana resultar VERDADEIRO. Observe a sintaxe logo abaixo.

```
ENQUANTO <expressão booleana> FAÇA
<instruções a serem executadas enquanto a expressão booleana resultar em VERDADEIRO>
FIM-ENQUANTO
```

Esse loop também é chamado de **loop pré-testado**, pois a expressão booleana é verificada antes da primeira execução. Se inicialmente a expressão já resultar em FALSE, então as instruções que estão dentro do bloco não serão executadas nenhuma vez.

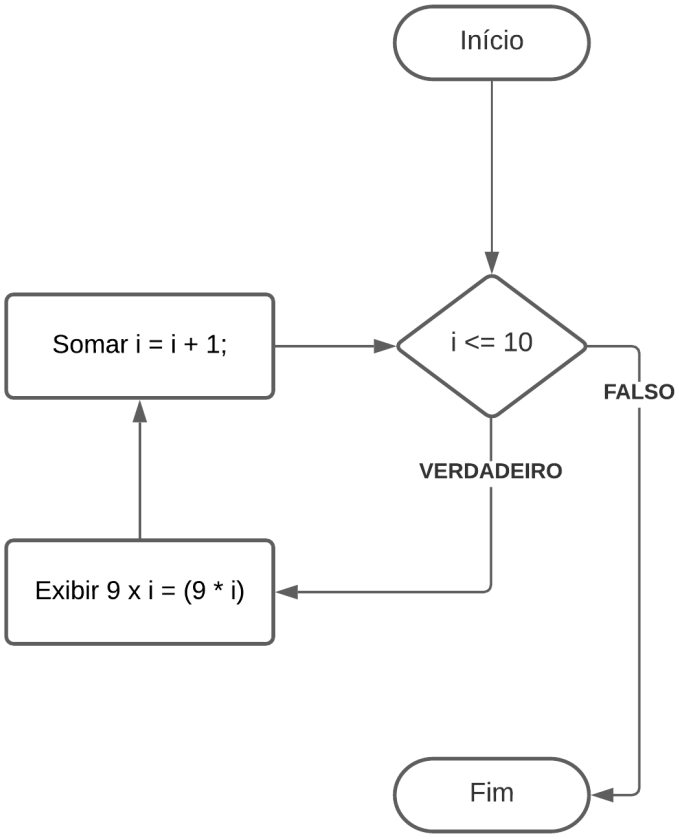
Vamos supor que queremos imprimir a tabuada do número 9. Como faríamos esse algoritmo? Observe o pseudocódigo.

```
Algoritmo tabuada_do_nove
Início
  var i: inteiro
  i ← 1
  Enquanto i ≤ 10 Faça
      Escreva "9 x i = (9 * i)";
      i ← i + 1;
  Fim-Enquanto
Fim-Algoritmo
```

Compreendeu o que está ocorrendo? Criamos uma variável **i** que recebe o valor 1. Enquanto **i** for menor ou igual a 10, nosso algoritmo vai escrever a tabuada. Para que o laço não se torne um **loop infinito**, precisamos incrementar o valor de **i** a cada passo. Se nós não incrementarmos o valor de **i**, ele sempre será 1 e a condição será TRUE para sempre (um laço infinito).

Outra coisa interessante é como o algoritmo sabe interpretar o valor de **i** dentro da instrução **Escreva**. Nós simplesmente colocamos o **i** ali e deixamos por conta do algoritmo, que ele pegue o valor atual da variável. Observe que estamos misturando o valor da variável com uma expressão literal [`"9 x i = (9 * i)";`]. Normalmente temos de realizar alguma conversão ou utilizar algum recurso da linguagem para que esse artifício funcione. Para um algoritmo digitado num editor de textos como o Word e que será apresentado apenas para a validação da ideia, você não precisa se preocupar com esse detalhe.

Como ficaria o fluxograma do algoritmo da tabuada? Vejamos.



Características Gerais

- Uma condição é avaliada no início.
- A avaliação da condição resulta em valores TRUE ou FALSE.
- Se o resultado da condição é FALSE, não é iniciada a repetição ou, caso esteja em execução, é encerrada a repetição.
- Se o resultado da condição é TRUE, é iniciada a repetição ou, caso esteja em execução, é reiniciada a execução das instruções dentro da Estrutura de Repetição.
- A avaliação da condição é sempre novamente realizada após a execução da última instrução dentro da estrutura de repetição.
- Se o resultado da condição é FALSE logo de primeira, o código não será executado nenhuma vez.
- Se estiver utilizando um contador para controlar o laço, este contador deve ser incrementado dentro do laço, do contrário teremos um laço infinito.

Estrutura de Repetição REPITA-ATÉ

O funcionamento da estrutura de repetição REPITA-ATÉ (em inglês REPEAT-UNTIL) é um pouco diferente da ENQUANTO-FAÇA. Ela também é chamada de FAÇA-ENQUANTO (Em inglês DO-WHILE). Observe sua sintaxe.

```
REPITA
<instruções a serem executadas até que a expressão booleana retorne VERDADEIRO>
ATÉ <expressão booleana>
```

Esse loop também é chamado de **loop pós-testado**. Isso significa que a verificação para repetir o LOOP é testada no final do bloco. Mesmo que inicialmente a expressão retorne FALSE, as instruções que estão dentro do bloco serão executadas pelo menos uma vez.

Outra coisa muito importante a se notar é que, além de ser pós-testada, esta estrutura testa o contrário do ENQUANTO-FAÇA. Na estrutura REPITA-ATÉ, as instruções do bloco são executadas repetidamente **enquanto a expressão booleana resultar FALSE**. A partir do momento que a expressão booleana resultar TRUE, o fluxo do algoritmo sairá do loop. O mesmo não vale para o FAÇA-ENQUANTO, como veremos mais a frente.

Vamos ver como fica nosso algoritmo da tabuada do 9 usando o REPITA-ATÉ.

```
Algoritmo tabuada_do_nove
Início
  var i: inteiro
  i ← 1
  Repita
      Escreva "9 x i = (9 * i)";
      i ← i + 1;
  Até i = 10
Fim-Algoritmo
```

Note as diferenças para a implementação realizada no ENQUANTO-FAÇA. A condição de parada lá era “enquanto i for menor ou igual 10”. Aqui a condição de parada é “até que i seja igual a 10”. Lá no ENQUANTO-FAÇA, se começarmos atribuindo o valor 11 à variável **i**, antes de começar o laço, nada vai acontecer. Se fizermos a mesma coisa aqui, teremos um loop infinito. Sério? Sim, pois se começarmos o laço com 11, ele vai incrementar o **i** a partir daí e ele nunca será igual a 10, ou seja, nunca chegará na condição de parada. Poderíamos modificar um pouco nosso algoritmo para evitar esse problema. Veja.

```
Algoritmo tabuada_do_nove
Início
  var i: inteiro
  i ← 1
  Repita
      Escreva "9 x i = (9 * i)";
      i ← i + 1;
  Até i >= 10
Fim-Algoritmo
```

Agora o programa funciona até que o **i** seja maior ou igual a 10. Neste caso, mesmo que a variável **i** comece com um valor maior, ela sairá do laço. No entanto, não importa o valor inicial que você fornece ao **i**, ele vai executar o bloco de código pelo menos uma vez!

Tem uma pegadinha aqui. Provavelmente você só vai perceber essa pegadinha quando tentar implementar esse laço em uma linguagem qualquer. O exemplo que mostramos acima é mais adaptável para uma linguagem como o Pascal ou Delphi, pois implementam o REPEAT-UNTIL. Já o Java, JavaScript, C, C++, C# e muitas outras implementam o DO-WHILE. E tem diferença? Vamos ver.

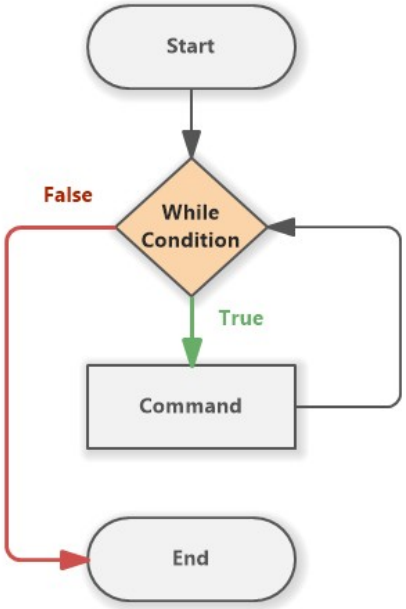
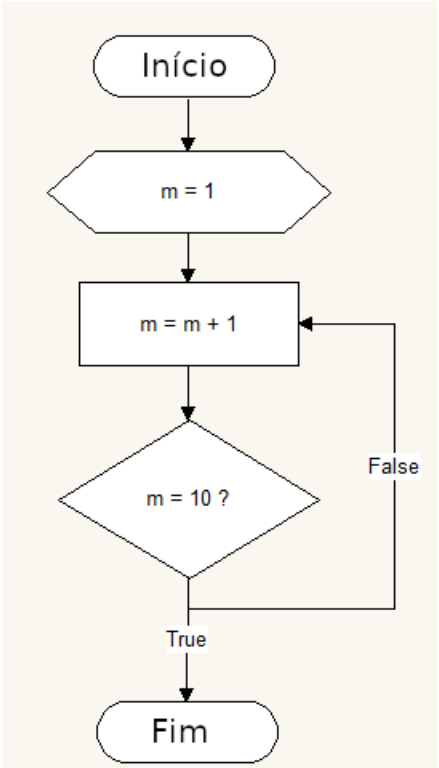
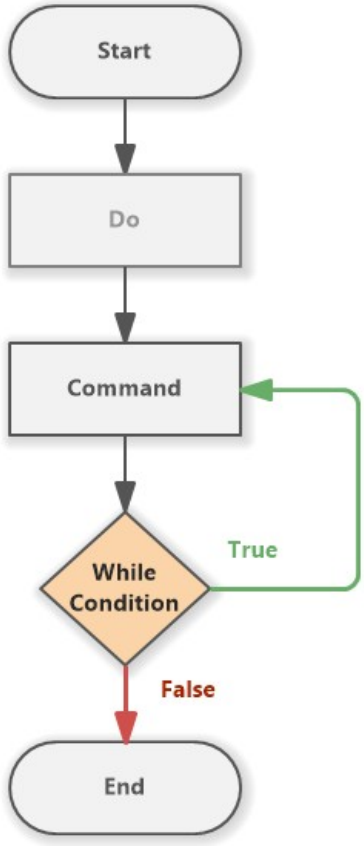
```
Algoritmo tabuada_do_nove
Início
  var i: inteiro
  i ← 1
  Faça
      Escreva "9 x i = (9 * i)";
      i ← i + 1;
  Enquanto i >= 10
Fim-Algoritmo
```

Nesse primeiro momento apenas alteramos de **Repita *para* Faça** e de **Até *para* Enquanto**. Observe que a lógica mudou. Agora o programa vai rodar enquanto **i** for maior ou igual a 10. Ele vai executar o código apenas uma vez e vai sair. E se utilizássemos nossa primeira estratégia de fazer o **i** igual a 10? Mesma coisa, ele vai executar o código apenas uma vez e vai sair. E agora? Devemos fazer o seguinte:

```
Algoritmo tabuada_do_nove
Início
  var i: inteiro
  i ← 1
  Faça
```

```
      Escreva "9 x i = (9 * i)";
      i ← i + 1;
  Enquanto i <= 10
Fim-Algoritmo
```

O teste fica diferente. Ele deve executar o código **enquanto *i* for menor ou igual a 10**. Agora sim vai funcionar. Perceba então que a estrutura do REPITA-ATÉ e do FAÇA-ENQUANTO é a mesma, ou seja, o teste é feito no final, porém, o teste em si será diferente para que funcione de acordo com a implementação da linguagem.

ENQUANTO-FAÇA (while)	REPITA-ATÉ (repeat…until)	FAÇA-ENQUANTO (do…while)
		

Aprenda muito bem como os laços funcionam! As estruturas de repetição são muito utilizadas em desenvolvimento de softwares. Entender como elas funcionam é muito importante para resolver problemas que precisam executar tarefas repetidas vezes. É algo rotineiro e você vai usar muitas e muitas vezes.

Mas ainda não acabou. Falta o PARA-FAÇA.

Estrutura de Repetição PARA-FAÇA

Essa é, provavelmente, a estrutura de repetição mais utilizada por todos os programadores. Quando criamos um algoritmo, muitas vezes já sabemos a quantidade de vezes que um laço vai se repetir. Por exemplo, imprima o nome de todos os alunos, sendo que temos 500 alunos. Neste caso, sabemos que o nosso laço será executado 500 vezes. Observe sua sintaxe.

```
PARA <contador> DE <valor inicial> ATE <valor final> [PASSO <valor de incremento>]
FAÇA
<instruções a serem executadas repetidamente até o <contador> atingir o valor final>
FIM-PARA
```

Esse é um **loop pré-definido**. Como assim? Nós sabemos desde o início quantas vezes ele será executado. Ué, mas lá no exemplo da tabuada eu também sabia quantas vezes o laço seria executado!

Sim, é verdade! Nós usamos um contador lá no exemplo da tabuada e já sabíamos quantas vezes o laço seria executado. Podemos usar aquela estrutura de repetição dessa maneira, mas o **PARA-FAÇA** foi criado especificamente para isso. E qual seria outra maneira de usar o ENQUANTO-FAÇA que não utilize um contador? Inúmeras! Mas vamos para um exemplo. Digamos que queremos pedir que o usuário adivinhe um número. O programa só vai parar quando o usuário adivinhar o número. Veja como fica com o ENQUANTO-FAÇA.

```
Algoritmo advinhe
Início
  var i, j: inteiro
  i ← 13
  Enquanto (j != i) Faça
      Escreva "Adivinhe o número que estou pensando - entre 1 e 20";
      Leia (j);
  Fim-Enquanto
  Escreva "Acertou mizerávi";
Fim-Algoritmo
```

Pronto. No programa acima não sabemos previamente quantas vezes o laço será executado. Só depende do usuário. O programa vai ficar pedindo para ele digitar o número até que ele informe o número correto, que neste caso é 13.

Muito bem, mas vamos então voltar para o nosso famoso exemplo da tabuada para dessa vez o implementarmos com o PARA-FAÇA. Observe.

```
Algoritmo tabuada_do_nove
Início
  PARA i DE 1 ATE 10 FAÇA
      Escreva "9 x i = (9 * i)";
  FIM-PARA
Fim-Algoritmo
```

Ficou bem menor o código não é mesmo? Bem melhor! Quando for implementar esse código numa linguagem de programação, novamente você deverá fazer pequenas adaptações para que funcione de acordo com a sintaxe da linguagem.

Vamos analisar novamente a sintaxe do FAÇA-PARA.

```
PARA <contador> DE <valor inicial> ATE <valor final> [PASSO <valor de incremento>]
FAÇA
<instruções a serem executadas repetidamente até o <contador> atingir o valor final>
FIM-PARA
```

Note que no exemplo da tabuada feito com o FAÇA-PARA nós não utilizamos a opção [PASSO <valor de incremento>]. Essa opção serve para informarmos para o laço quantas vezes ele deve incrementar o contador cada vez que o laço for executado. O padrão desse incremento é 1 e foi por isso que não informamos nada ali.

Vamos imaginar que queremos imprimir todos os números pares de 1 até 20. Como faríamos? Observe.

```
Algoritmo pares
Início
  PARA i DE 2 ATE 20 PASSO 2 FAÇA
      Escreva i;
  FIM-PARA
Fim-Algoritmo
```

Viu só como usamos o PASSO ali? Agora ele vai incrementar o **i** de 2 em 2. Dessa maneira começamos o programa com 2 e vamos imprimindo todos os números até chegar ao 20. Mas quando ele for incrementar o **i**, que no início é 2, ele vai passar para 4, depois para 6 e assim por diante, pois definimos o PASSO como 2.